THE REMARKABLE EFFECTIVENESS OF COMBINING POLICY AND VALUE NETWORKS IN A*-BASED DEEP RL FOR AI PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite the tremendous success of applying traditional backtrack-style combinatorial search methods in various NP-complete domains such as SAT and CSP as well as using deep reinforcement learning (RL) to tackle two-player games such as Go, PSPACE-hard AI planning has remained out of reach for current AI planning systems. Even carefully designed domain-specific solvers fail quickly due to the exponential combinatorial search space on hard instances. Recent work based on deep learning guided search algorithms that combine traditional search-based methods, such as A* and MCTS search, with deep neural networks' heuristic prediction has shown promising progress. These methods can solve a significant number of hard planning instances beyond specialized solvers. To better understand why these approaches work we study the interplay of the policy and value networks in A*-based deep RL and show the surprising effectiveness of the policy network, further enhanced by the value network, as a guiding heuristic for A*. To further understand the phenomena, we study the cost distributions of deep planners and found that planning instances can have heavy-tailed runtime distributions, with tails both on the right-hand and left-hand sides. In particular, for the first time, we show the existence of heavy *left* tails and propose a theoretical model that can explain the appearance of these tails. We provide extensive experimental data supporting our model. The experiments show the critical role of the policy network as a powerful heuristic guiding A*, which can lead to left tails with polynomial scaling by avoiding exploring exponential size sub-trees early on in the search. Our results also demonstrate the importance of random restart strategies, as are widely used in traditional combinatorial solvers, for deep reinforcement learning and deep AI planning systems to avoid left and right heavy tails.

1 INTRODUCTION

Combinatorial search is a key domain for artificial intelligence. Unfortunately, combinatorial domains usually have intractable theoretical complexity, such as NP-complete, PSPACE-complete, or even undecidable. In the past few decades, we have observed tremendous progress for practical problem solving in NP-hard domains with wide applicability in, for example, circuit design (Hong et al., 2010), hardware verification (Gupta et al., 2006), and mathematical discovery (Konev & Lisitsa, 2014). Boolean Satisfiability (SAT) solvers based on conflict-driven clause learning (CDCL) can solve instances with thousands of variables and clauses in seconds, which demonstrates surprising scaling performance in spite of SAT being an NP-complete task (Silva & Sakallah, 2003).

In contrast, practical combinatorial search in PSPACE-hard domains has remained a significant practical challenge. PSPACE-hard problems can be generally divided into two main categories: *two-player games*, such as Go (Lichtenstein & Sipser, 1980), Chess (Fraenkel & Lichtenstein, 1981), Amazons (Hearn, 2005), and *single-agent planning problems* (also known as combinatorial puzzles), such as problems captured by the Planning Domain Definition Language (PDDL) (Bylander, 1994) and Sokoban (Culberson, 1997). Determining the satisfiability of Quantified Boolean Formulas (QBF), the canonical example of a PSPACE-complete problem, can also be viewed as a game between the existential player and the universal player. Recent achievements in the deep learning community inspired an approach of augmenting Monte Carlo tree search (MCTS) with neural net-

works' heuristic predictions. AlphaGo (Silver et al., 2016) is the first Go software to outperform professional players in 2016, and its newer and more general version AlphaZero (Silver et al., 2017) can achieve, *tabula rasa*, superhuman performance in many other challenging game domains, such as Chess and Shogi.

These advances naturally raise the question of whether the success in game domains can be transferred to PSPACE-hard planning domains. A key challenge is that the Monte Carlo nature of the search algorithm, whose output may be incorrect with a certain (typically small) probability, is not well-suited for hard planning tasks, because they often require intricate action sequences with hundreds of steps and any deviation can lead to a "dead-end" state without a path to the goal.

To address this issue, a more systematic and complete search, e.g., using an A* style method is required. In fact, deep learning guided search algorithms that combine traditional search-based methods such as A* with deep neural networks' heuristic prediction have shown promising progress. These methods can solve a significant number of hard planning instances that specialized solvers cannot solve. To gain a better understanding of these approaches, we study the interplay of the *policy* and *value networks* in A*-based deep RL. Our experiments will show the surprising effectiveness of the policy network, further enhanced by the value network, as a guiding heuristic.

To further understand the phenomena, we also study the cost distributions of deep RL planners. Specifically, we explore and generalize the cost distribution profiles of search methods from NP-hard domains to PSPACE-hard planning domains, and show heavy-tailed cost distributions exist ubiquitously among planning instances. For NP domains, the maximum depth of the search tree and the branching factors are polynomially bounded. As a result, search methods are usually backtrack-style with powerful heuristics and branch cutting rules augmented. In contrast, planning methods relying on, e.g., A* search, maintain a search frontier of nodes (or states) to be explored and use domain knowledge to select the next node for expansion. Agostinelli et al. (2019) found that, for combinatorial planning problems, MCTS has relatively long runtimes and often produces solutions many moves longer compared with A* search. We therefore adopt (weighted) A* search as the basic search framework and use a deep neural network to provide search guidance.

Randomized combinatorial solvers use various techniques, such as randomized tie-breaking and random variable ordering, to carefully inject a *controlled amount* of randomization into a complete, deterministic, backtrack-style search procedure (Crawford & Baker, 1994; Bresina, 1996; Gomes et al., 1998). The randomization step requires careful engineering and analysis of the solver since excess randomness can hamper the effectiveness of random restart. In our approach, we found that *uncertainty-aware networks* (Huang et al., 2017; Chua et al., 2018; Sedlmeier et al., 2019; Malinin & Gales, 2018) can provide just the right amount of controllable randomness into a deterministic planning algorithm.

Our results show that many planning instances have heavy-tailed runtime distributions, with tails both on the right-hand and left-hand sides. Such heavy-tailed distributions have been observed before in NP-hard domains, such as SAT and CSP (Gomes et al., 1997; 1998) but not in PSPACE settings. Moreover, for the first time, we show the existence of heavy *left* tails and propose a theoretical model that can explain the appearance of these tails. We provide extensive experimental data supporting our model. The experiments show the critical role of the policy network as a powerful heuristics guiding A*, which can lead to left tails with polynomial scaling by avoiding exploring exponential size sub-trees early on in the search.

Our results also demonstrate the importance of *random restart strategies*, as are widely used in traditional combinatorial solvers, for deep reinforcement learning and deep AI planning systems to avoid left and right heavy tails.

In summary, our overall contributions are as follows:

1. We study *the interplay of the policy and value networks* in A*-based deep RL. Our experiments show *the surprising effectiveness of the policy network,* further enhanced by the value network, as a guiding heuristic for A*.

2. We identify *heavy-tailed runtime distributions* of PSPACE-hard planning problems and propose a series of distribution-independent statistics to quantify the heaviness of tails and effectiveness of random restart. *For the first time*, we show and extensively study *heavy*

left tails from experiment data, introduce an *abstract tree search model* with *critical nodes*, and formally show how heavy left tails can arise during the search.

3. The tails on the left of the runtime distribution are explained by the critical role of the policy network as a guiding heuristic. Polynomial runtime scaling can occur because the policy network helps avoid exploring exponential size sub-trees early on in the search. Figure 3 visualizes the phenomena by revealing a small set of critical nodes in the early part of the search space.
4. We show the importance of using *uncertainty aware* networks in the planning domain and how it can add a controllable amount of randomness to a backtrack-style solver.
5. We show how a *restart strategy* can improve the deep RL planner effectiveness. In particular, our experiments show how for a given search budget, there is an optimal restart strategy. For larger budgets, more frequent restarts are most effective.

2 Background and Related Work

Sokoban as a planning domain Sokoban is a PSPACE-complete puzzle whose goal is to push a set of boxes into the same number of goal squares in a grid with walls (Culberson, 1997). It is demonstrated as a canonical planning problem for AI, more complex than the others, and very challenging even when significant human domain knowledge is provided (Fern et al., 2011; Junghanns & Schaeffer, 2001). Due to its general search structure and hardness, we use Sokoban as our background domain throughout the paper.

Optimal speedup of Las Vegas algorithms Let $\mathcal{A}$ be a randomized algorithm that always outputs the correct answer when it halts but whose running time is a random variable $p_{\mathcal{A}} : \mathbb{Z}^+_\infty \rightarrow \mathbb{R}^+_0$. Luby et al. (1993) prove that when we have full knowledge about the distribution $p_{\mathcal{A}}$, the optimal strategy that achieves the minimum expected time required to obtain an output from $\mathcal{A}$ is to repeatedly run $\mathcal{A}$ for the same amount of time $t_{\mathcal{A}}$ until it halts. To calculate $t_{\mathcal{A}}$, let

$$l(t) = \frac{1}{\sum_{x \leq t} p(x)}(t - \sum_{x<t} \sum_{y \leq x} p(y))$$

be the expected halting time of repeatedly running $\mathcal{A}$ with time limit t. Define $l_{\mathcal{A}} = \inf_{t<\infty} l(t)$ and $l_{\mathcal{A}}$ is finite for any non-trivial distribution p, i.e., $p(\infty) < 1$. Let $t_{\mathcal{A}}$ be any finite value of t such that $l(t) = l_{\mathcal{A}}$, if such a value exists, or $t_{\mathcal{A}} = \infty$ otherwise. Luby et al. (1993) also show that when $p_{\mathcal{A}}$ is unknown, a universal strategy that runs $\mathcal{A}$ for time

$$1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 16, ...$$ [1]

can achieve estimated halting time $O(l_{\mathcal{A}} \log(l_{\mathcal{A}}))$ for any randomized algorithm $\mathcal{A}$. This bound is optimal among all universal strategies.

Heavy-tailed distributions of randomized search Gomes et al. (2000) observe heavy-tailed behavior on the right-hand side of experiments on SAT and CSP. Heavy right tails occur when most runs halt in polynomial time while a non-negligible fraction of extremely long runs makes the average running time exponential. On the contrary, heavy left tails occur when a non-negligible fraction of relatively short runs exists despite most of the remaining runs require exponential time. A rapid restart strategy can take advantage of such short runs, significantly reducing the expected halting time. Figure 1 shows the comparison of heavy tails and exponential tails on various statistics. As shown in the figure, for heavy-tailed distributions, using random restart based on the optimal $t_{\mathcal{A}}$ or the universal strategy can achieve orders of magnitude shorten running time compared with the mean of the runtime sample, both for left-hand and right-hand tails. The difference between the sample mean and restart is *insignificant* for exponential tails.

Uncertainty-aware network There is a line of research on the uncertainty of deep neural networks to reduce test error, provide confidence estimate, and improve model-based reinforcement learning (Huang et al., 2017; Chua et al., 2018; Sedlmeier et al., 2019; Malinin & Gales, 2018). Our method utilizes test-time dropout networks as heuristic functions to introduce a controlled amount of randomization to deterministic search engines (Gal & Ghahramani, 2016). The uncertainty comes

[1] https://oeis.org/A182105

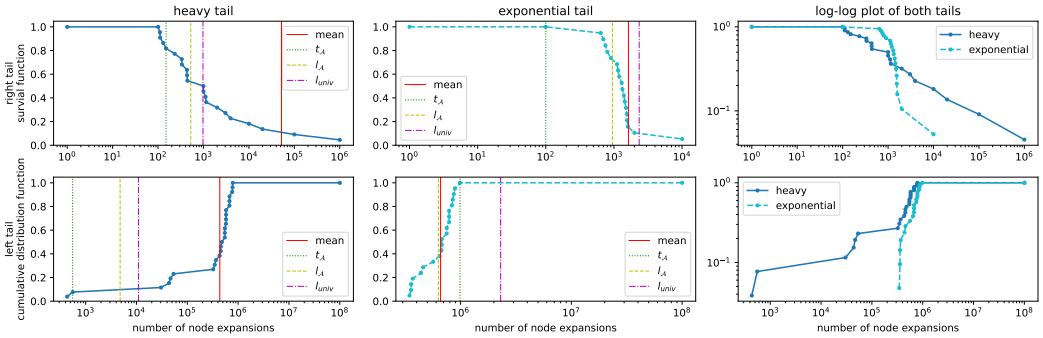


Figure 1: Each subplot shows multiple A* runs with the Monte Carlo dropout on one Sokoban instance. We compare the runtime sample mean, optimal restart time $t_{\mathcal{A}}$ of the discrete uniform distribution on the runtime sample, optimal estimated halting time $l_{\mathcal{A}}$, and the estimated halting time l_{univ} using the universal strategy.

from two main sources: (1) the distributional mismatch between the training and test states; (2) noises in the training data since the remaining distances (found by specialized solvers) are usually not optimal.

Policy-guided heuristic A* search Several works augment A* search with neural networks that estimate the cost-to-goal distance, policy distribution, or both (Agostinelli et al., 2019; Orseau et al., 2018; Orseau & Lelis, 2021). Orseau & Lelis (2021) add path probability $\pi(n)$ to the minimizing term of A* search, where $\pi(n) = p(s_1|s_0)p(s_2|s_1)\cdots p(s_l|s_{l-1})$ is the cumulative product of networks' policy predictions among the path from the root to n. We found that for uncertainty-aware networks whose prediction is well-calibrated, $\pi(n)$ converges to zero exponentially fast as the depth of n grows. The search quickly degenerates into BFS when goals require a long plan since shallow nodes are always preferred by A* based on $\pi(n)$. Our method modifies the term $\pi(n)$ to $p(s_l|s_{l-1})$ to avoid exponential growth of the minimizing term of A* as the depth of nodes grows.

3 Formal Framework

The study of the complexity and practical performance of search methods is greatly hampered by the difficulty in collecting real data. As an alternative, researchers heavily resort to procedurally generated instances or highly structured problem domains (Taylor & Parberry, 2011; Guez et al., 2018). The randomly generated instances lack sufficient structure and their underlying combinatorial search space is, in some sense, too regular. To bridge this gap, we collect planning instances from various sources with structural differences. All these instances have practical interest and serve as a benchmark not only for automated solvers but also for humans to solve.

3.1 Data Preparation

Table 1: Comparison with previous DRL works on Sokoban planning domains

Related works	avg width	avg height	avg size	avg boxes
I2As (Racanière et al., 2017)	10	10	100	4
LevinTS (Orseau et al., 2018)	10	10	100	4
PHS (Orseau & Lelis, 2021)	10	10	100	4
Feng et al. (2020b)	13	19	247	16
Our setting	12.0	13.6	183.5	20.2

We collect all Sokoban instances from the SokobanWiki website[2], resulting in 10871 instances in total. A subset called Large Test Suite with 3272 instances serves as the default benchmark test set for various modern specialized Sokoban solvers (Shoham & Elidan, 2021). All these instances

[2]http://sokobano.de/en/levels.php

are created by different human authors in the past few decades and have great differences in the underlying structure. The dataset is *orders of magnitude larger* than ones used in previous works on deep reinforcement learning and provides a great challenge for deep heuristic learning. See Table 1 for Sokoban board statistics compared with previous works. Notice that the difficulty of Sokoban grows exponentially as the number of boxes increases.

To generate supervised learning signals for networks, we run Sokolution[3], which is a state-of-the-art domain-specialized solver for Sokoban, to compute ground truth solution paths. Sokolution can solve 8272 out of 10871 total instances given a 10-minute time limit. For each found plan path $(s_0, a_1, s_1, ..., a_n, s_n)$, we generate policy label $p_{s_i} = a_{i+1}$ and remaining distance label $v_{s_i} = n-i$ as the supervised learning signals for deep networks. We further divide the solved 8272 instances into a training set (7435 instances) and a test set (827 instances) and make the remaining unsolved 2609 instances as the hard set to further study the cost distribution of instances that are way harder than the training instances.

Feng et al. (2020a) use "push" as a single move action to achieve the state-of-the-art performance of deep RL in the Sokoban domain. In this work, we use the more elementary action "move". Notice that the "push" action requires more domain knowledge of Sokoban, e.g., the framework needs to compute the reachable cells from the current player location and decides which boxes are pushable. "move" consists of four actions: upward, downward, rightward, and downward. The framework only needs to detect whether the four adjacent cells are empty, or whether the box is pushable if one occupies an adjacent cell. Using "move" as basic actions will generate plans that are usually orders of magnitude longer than ones generated with "push" actions. For the testing dataset in the paper, the majority of instances have solution length contains hundreds or even thousands of moves, which provides a great challenge for the AI agent.

3.2 POLICY-GUIDED A* SEARCH

A* is an informed search algorithm or a best-first search. Starting from a specific start state, its goal is to find a plan/path to one of the goal states with potentially smallest cost (least cumulative cost, least plan length, shortest running time, etc). Specifically, A* gradually enlarges the current search graph by consecutively expand a new node which minimizes a cost function $f(n) = g(n) + h(n)$, where n is the next node to expand, $g(n)$ is the minimum cost of the path (if multiple paths exist) from the start node to n, and $h(n)$ is a heuristic function that estimates the minimum remaining cost to one of the goals. Traditional A* with admissible and consistent heuristic functions can guarantee the first path found to a goal state has the optimal cost. When augmenting A* with deep networks' heuristic $h(n)$, we usually loosen the optimality constraint since admissibility and consistency of heuristic functions can no longer be satisfied. We use the path length as the cost of A* search so that $g(n)$ is the depth of node n.

Orseau & Lelis (2021) propose Policy-guided Heuristic Search (PHS) to add a policy term to the cost function $f(n)$. Specifically, they modify the cost function to

$$f(n) = \frac{g(n) + h_\theta(n)}{\pi_\theta(n)}, \pi_\theta(n) = p_\theta(s_1|s_0) \cdots p_\theta(s_m|s_{m-1}),$$

where $(s_0, ..., s_m)$ is the path from the root to n. We found that PHS will quickly degenerate into BFS on hard planning instances for well-calibrated networks. We resolve this issue by replacing the cost function with

$$f(n) = \frac{g(n) + h_\theta(n)}{\max_{x \in \text{parents of } n} p_\theta(n|x)}.$$

PHS* is a significantly improved version of PHS that uses the geometric mean of probabilities among the path from the root to n instead of $\pi_\theta(n)$. We found simply using $\max_{x \in \text{parents of } n} p_\theta(n|x)$ can consistently achieve better performance in Sokoban.

We train a network with a dropout probability 30% on the remaining distance and policy labels from the training set. For evaluating search cost distributions we test the model on all the training, test, and hard dataset. We found the combinatorial search is so challenging that the model cannot reliably solve planning instances in the training set even has been trained on ground truth labels from it.

[3]http://codeanalysis.fr/sokoban

As the evaluation time of deep networks heavily depends on hardware, we use the number of expanded nodes of A* as the runtime cost instead of the actual running time. We consider duplicate board detection so that various nodes with the same board will merge into a single node. The original A* search puts a previously expanded node n back to the open set when its $g(n)$ (the distance from the start state to n) is updated to a smaller value. To increase the algorithm's efficiency and improve the evaluation metric, each time a node n is popped out from the open set (frontier nodes of the search), we do a local Shortest Path Faster Algorithm (SPFA) starting from n to ensure that no relaxation exists in the closed set at any stage of A* search, i.e., $g(u) + 1 \geq g(v)$ for each pair of adjacent nodes u, v while u is in the closed set.

4 THE IMBALANCED TREE MODEL

In this section, we describe the proposed abstract model that can explain left-hand heavy tails. We mathematically prove how heavy tails occur and random restart helps, and empirically justify the model with the experiment data. See Figure 2 for the description for both models.

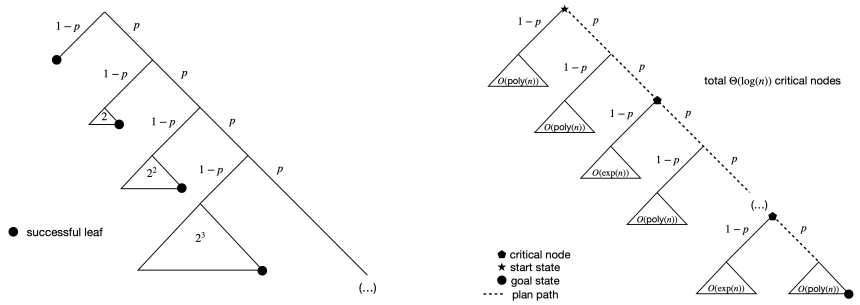


Figure 2: Left panel: imbalanced tree model for right-hand heavy tails. Right panel: our proposed model for left-hand heavy tails.

4.1 HEAVY TAILS ON THE RIGHT

Chen et al. (2001) create the imbalanced tree models for SAT and CSP search. The search has an exponentially lower possibility $(1-p)p^m$ to explore exponentially larger 2^m search space. Specifically, randomized search has an infinite mean and infinite variance with $p \geq 0.5$. With $0.25 < p < 0.5$, the search has a finite mean but infinite variance. And finally, with $p \leq 0.25$, the search has both a finite mean and finite variance. The right tails of the model follow the Pareto-Lévy form: $P(X > x) \sim Cx^{-\alpha}, x > 0$.

4.2 HEAVY TAILS ON THE LEFT

Search models for planning problems are quite different from ones for SAT and CSP. For NP problems, the number of unassigned variables is fixed with $O(n)$ and the search can assign these variables in any order. For planning domains, we consider finding a path from the start state to a goal state with potentially maximum exponential length. Our proposed model assumes the existence of $O(log(n))$ **critical nodes** among the plan from which a wrong child node selected by A* will result in extra exponential search space. As will be shown in follows, our model does not depend on the actual choice of p as long as p is a constant value in range $(0, 1)$.

Theorem 4.1 *The possibility that $\mathcal{A}$ has exponential runtime on input of size n converges to 1 as n goes infinite.*

Theorem 4.2 *The optimal restart time $t_{\mathcal{A}}$, optimal expected halting time $l_{\mathcal{A}}$, and expected halting time using the universal strategy l_{univ} are all $O(poly(n))$.*

Theorem 4.1 states the average running time of the model is exponential on n, and Theorem 4.2 shows that the cost distribution has only polynomial restart estimated running time. These two theorems combined show the occurrence of left-hand heavy tails.

Here is an intuitive explanation of the abstract model. For the majority of the nodes on the solution plan, the deep neural network either provides an accuracy heuristic to choose the right child node or makes a small error of preferring a wrong child node. As long as the error is small A* search can recover from it with extra polynomial steps since A*'s cost function penalizes deeper nodes. For $O(\log(n))$ critical nodes, the error of heuristic estimation of the deep network is so large that exponential search is needed to jump out from the local search space. We empirically found that in the search graph of A*, the majority of nodes **do not** branch — heuristics provided by the network is accurate enough to prefer the right child node with high confidence.

The proposed model has a close relation to the backdoor model of NP search. To explain why SAT solvers work so well in practical instances, Hoffmann et al. (2006) examine various benchmarks and identify that for most practically solvable SAT instances, after assigning values to *logarithmic* variables, the remaining problem instance quickly becomes polynomially solvable by propagation rules. This result illuminates the prototypical patterns of the structure causing the empirical behavior observed in the International Planning Competitions benchmarks.

As for PSPACE-hard planning domains, we also found that the majority of decisions among the plan, i.e., choosing which child node to expand next, has only a polynomial cost. This implies the network either is very accurate in predicting the right child, or A* only needs a shallow search to recover from a wrong expansion. This can also explain why previous DRL models can be so effective in solving hard Sokoban instances that are out of reach for modern specialized solvers (Feng et al., 2020a;b).

5 EXPERIMENTS

5.1 STRUCTURE OF REAL SEARCH GRAPHS

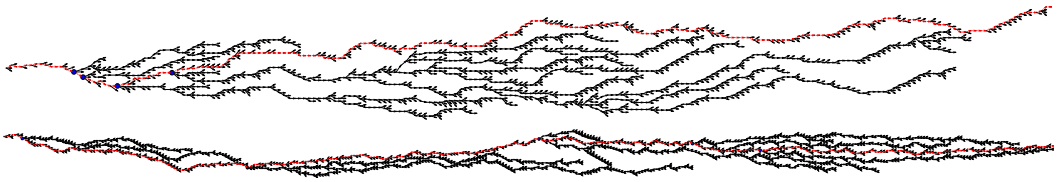


Figure 3: **Two typical deep A* search graphs.** Without affecting demonstration, we remove extra graph edges to form a spanning tree for clearer illustration. The search graph is built from the left to the right. Red dashed path is the solution plan leading form the start state to one goal state, while blue circles are *critical nodes*, which have exponential sub-search tree without a close goal state.

Figure 3 shows two typical search graphs generated by deep A* search. We found that the majority of nodes actually **do not** incur search. Deep networks' heuristic either direct leads A* to the correct child node, or only a very small wrong sub-tree (less than 5 nodes) is explored. *Critical nodes* (labeled as blue circles) are extremely rare in the search graph. However, when encounters such critical nodes, the search engine needs to explore a large sub-tree before noticing no close goal exists in the current search space.

Whether AI planning systems can find **routine macro action**, i.e., a sequence of algorithmic actions to perform a sub-goal, has a great interest for researchers. To make a long plan, e.g., prove a hard mathematical theorem, humans usually only make some *critical choices* of lemmas and schemes, and fill the remaining part of the proof without extra reasoning. Indeed the number of required *crucial intermediate lemmas* to prove a mathematical theorem is quite small, even for challenging open problems. However, extensive and profound reasoning, search, and enumeration are needed to find such lemmas. The small number of critical lemmas compared with the long proof length reflects the property of such search graphs. In our experiment setting, we use "move" as the basic action unit. To perform a real "push", the planner is asked to compute the reachable cells from the player's location and calculate the shortest path to the cell adjacent to the box waiting to push. Such a long sequence of moves before performing an actual push can be viewed as a **routine**. As shown in Figure 3, the deep A* search can perform a long chain of moves without any search, which is a clear demonstration that routine macro actions are **implicitly** learned as a part of heuristic functions.

5.2 EFFECTIVENESS OF POLICY AND VALUE NETWORKS

Table 2: Solving percentage on the test set with various cost functions of A* search, depending on depth $d(n)$, remaining distance $h_\theta(n)$, action probability $p_\theta(n|\text{par}(n))$, and cumulative path probability $\pi_\theta(n)$. Columns represent different search budget (number of total node expansions).

Method	A* Cost function	Number of total node expansions					
		1K	2K	4K	8K	16K	32K
No Policy							
BFS	d	0.32%	1.28%	1.92%	3.21%	6.73%	11.2%
Pure Distance	h	4.17%	8.01%	12.5%	15.1%	19.2%	19.6%
Vanilla A*	$d + h$	5.77%	10.3%	13.5%	19.9%	21.5%	25%
Weighted A*	$d + 2.0 \cdot h$	6.09%	8.97%	14.7%	17.6%	19.6%	22.8%
Weighted A*	$d + 4.0 \cdot h$	5.45%	9.62%	13.1%	17.3%	19.9%	21.8%
PHS variants							
PHS	$(d + h)/\pi$	15.7%	19.9%	23.7%	26.3%	29.2%	31.4%
PHS*	$(d + h)/\pi^{1+h/d}$	28.5%	31.4%	38.5%	40.1%	44.9%	46.2%
PHS* Simplified	$1/\pi^{1+h/d}$	28.5%	30.8%	39.1%	40.7%	45.5%	45.5%
Geometric Mean	$1/\pi^{1/d}$	24.0%	28.2%	30.1%	32.4%	34.0%	36.2%
Pure Policy	$1/\pi$	14.4%	18.6%	22.4%	25.3%	28.2%	30.1%
Ours							
Pure Policy	$1/p$	28.2%	32.1%	36.9%	40.4%	42.9%	44.6%
Policy + BFS	d/p	15.1%	18.9%	23.4%	26.0%	28.2%	30.8%
Policy + Distance	h/p	31.7%	32.4%	37.5%	38.8%	41.0%	41.3%
Policy + A*	$(d + h)/p$	**32.4%**	**34.3%**	38.8%	**43.3%**	**46.2%**	**50.0%**
Policy + Weighted	$(d + 2.0 \cdot h)/p$	31.7%	34.0%	**39.4%**	42.0%	45.8%	48.1%
Policy + Weighted	$(d + 4.0 \cdot h)/p$	31.1%	32.7%	37.5%	40.1%	43.3%	44.9%

Table 2 shows various experiment results on different choices of the A* cost function. We summarize our crucial findings as follows:

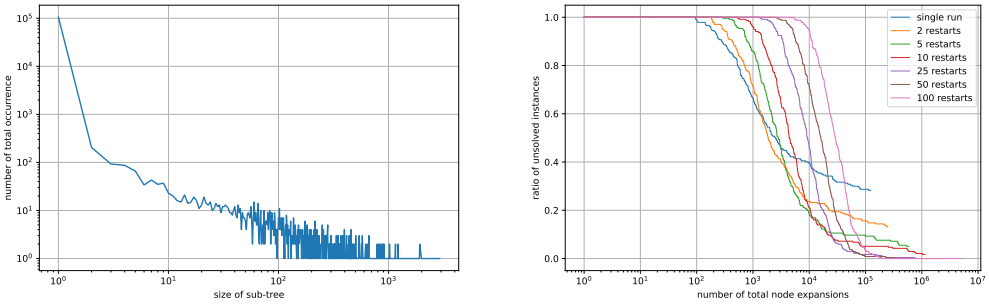
(1) **The policy heuristics has a significantly larger impact than the distance heuristics**. Specifically, Pure Policy can achieve close performance to our best strategy, while Pure Distance is significantly worse. Our intuition is that the distance heuristics is trained from solution paths so it possesses strong locality property, i.e., neighbor states have similar remaining distance predictions. For Sokoban where ubiquitous dead-end states exist, it's challenging for Pure Distance to distinguish the correct child state from significantly worse child states.

(2) **Adding the depth term helps increase the solving rate**. In theory, the depth term does not contain extra heuristic information about the search space, and bringing in this term is usually to enforce **shorter** plans. In the experiment, we found the depth term can also increase the solving ratio. The intuition is the distance heuristics sometimes can greatly underestimate a bad state during the search and force A* to explore a large search space beneath it. Adding the depth term can help the search jump out from that search space faster.

(3) **Weighted A* does not help**. In contrast, traditional A* frameworks use the weighted version to help increase the solving ratio in trade of the quality of solution length. Our explanation is traditional A* uses a human-designed admissible heuristic which serves as a **lower bound** of the ground truth distance. However, deep neural networks learn the distance function from solved planning instances so that the learned $h_\theta(n)$ is a **upper bound** of the ground truth, which naturally increases the impact of the distance heuristics over the depth $d(n)$.

5.3 EXPERIMENT DATA FOR THE ABSTRACT MODEL

To better illustrate the occurrence probability of different sizes of sub-search space, we randomly sample 300 solved instances and count the number of dead sub-trees (no close goal exists) for each size. As shown in Figure 4a, as the size of the tree grows exponentially, the number of sub-trees also decreases exponentially fast. The experiment data confirms our abstract model that critical nodes



(a) Subtree Size (b) Unsolved Ratio

Figure 4: (a) The total number of sub-trees of 300 randomly sampled instances v.s. the tree size (in log-log scale). (b) Unsolved fraction of instances with or without restarts. Given a fixed amount of search budget (number of nodes allowed to expand), n-restart means to evenly divide the search budget into n individual runs and union their solved instance sets as the final result.

which incur extra exponentially large sub-search space have logarithmic occurrence. This is clear evidence about how left-hand heavy tails occur in practical planning solving.

Hoffmann et al. (2006) has given an explanation to STRIPS-style planning with backdoor models — after finding and assigning logarithmic variables the remaining problem solving becomes polynomial. In contrast, finding these critical nodes is not necessary for expand-style search algorithms. Heavy left tails are not only caused by the underlying structure of practical instances but also affected by the mysterious generalization ability of deep neural networks. It is widely believed and experimentally confirmed that with polynomially increasing model size, deep networks can achieve "exponential" scaling power to unseen states. It is an interesting future research direction to understand the surprising scaling performance of various heuristic methods, from Conflict-driven clause learning (CDCL) for SAT solving, to deep networks for various combinatorial domains.

5.4 Solving more instances with random restart

The theory of heavy-tailed cost distribution suggests that a sequence of short runs instead of a single long run may make better use of a fixed amount of computational budget. We explore this idea by considering a fixed number of total expanded nodes allowed during A* search. Figure 4b shows the expected probability of *not* solving the instances for the train, test, and hard dataset. The failure rate of a single run drops the fastest for a small amount of computational budget. With more total expanded nodes, the random restart method gradually achieves better performance. Specifically, to solve more instances, you need to increase the total number of compute cycles. When doing so, the figure shows there comes a point where more frequent restarts are more effective. For example, with a budget of around 2,000 nodes, a policy with 2 restarts becomes more effective than no restarts (orange curve crosses blue curve). At around 5,000 nodes, 5-restart becomes more effective than 2-restart (green curve crosses blue curve), etc. So, to solve a larger fraction of hard instances, more frequent search restarts become more effective.

6 Conclusion

We studied the use of policy (action selection) and value (remaining distance estimate) function information as well as randomization methods for solving hard planning instances using A* search. Our experiments show the remarkable effectiveness of the policy network and random restarts for deep A* search. The value network provides additional global search guidance.

We showed that *uncertainty aware* networks provide an effective way to introduce randomization into the search process leading to increased efficiency. Our runtime distribution results show *heavy-tailed distributions* with tails on both the left-hand and right-hand sides. Left tails have not been observed in combinatorial search before. We also introduced an abstract computational model that explains heavy left tails. Finally, we showed how *random restarts* can improve overall search effectiveness. With larger search budgets to solve harder instances, restarts are increasingly effective.

REFERENCES

Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8):356–363, 2019.

John L Bresina. Heuristic-biased stochastic sampling. In *AAAI/IAAI, Vol. 1*, pp. 271–278, 1996.

Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

Hubie Chen, Carla Gomes, and Bart Selman. Formal models of heavy-tailed behavior in combinatorial search. In *International Conference on Principles and Practice of Constraint Programming*, pp. 408–421. Springer, 2001.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

James M Crawford and Andrew B Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Aaai*, volume 2, pp. 1092–1097, 1994.

Joseph Culberson. Sokoban is pspace-complete. 1997.

Dieqiao Feng, Carla P Gomes, and Bart Selman. A novel automated curriculum strategy to solve hard sokoban planning instances. *Advances in Neural Information Processing Systems*, 33, 2020a.

Dieqiao Feng, Carla P Gomes, and Bart Selman. Solving hard ai planning instances using curriculum-driven deep reinforcement learning. *arXiv preprint arXiv:2006.02689*, 2020b.

Alan Fern, Roni Khardon, and Prasad Tadepalli. The first learning track of the international planning competition. *Machine Learning*, 84(1-2):81–107, 2011.

Aviezri S Fraenkel and David Lichtenstein. Computing a perfect strategy for n$\times$ n chess requires time exponential in n. In *International Colloquium on Automata, Languages, and Programming*, pp. 278–293. Springer, 1981.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

Carla P Gomes, Bart Selman, and Nuno Crato. Heavy-tailed distributions in combinatorial search. In *International Conference on Principles and Practice of Constraint Programming*, pp. 121–135. Springer, 1997.

Carla P Gomes, Bart Selman, Henry Kautz, et al. Boosting combinatorial search through randomization. *AAAI/IAAI*, 98:431–437, 1998.

Carla P Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of automated reasoning*, 24(1):67–100, 2000.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. An investigation of model-free planning: boxoban levels, 2018.

Aarti Gupta, Malay K Ganai, and Chao Wang. Sat-based verification methods and applications in hardware verification. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pp. 108–143. Springer, 2006.

Robert A Hearn. Amazons is pspace-complete. *arXiv preprint cs/0502013*, 2005.

Jörg Hoffmann, Carla P Gomes, and Bart Selman. Structure and problem hardness: Goal asymmetry and dpll proofs in sat-based planning. In *ICAPS*, pp. 284–293, 2006.

Ted Hong, Yanjing Li, Sung-Boem Park, Diana Mui, David Lin, Ziyad Abdel Kaleq, Nagib Hakim, Helia Naeimi, Donald S Gardner, and Subhasish Mitra. Qed: Quick error detection tests for effective post-silicon validation. In *2010 IEEE International Test Conference*, pp. 1–10. IEEE, 2010.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

Andreas Junghanns and Jonathan Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129(1-2):219–251, 2001.

Boris Konev and Alexei Lisitsa. A sat attack on the erdős discrepancy conjecture. In *International conference on theory and applications of satisfiability testing*, pp. 219–226. Springer, 2014.

David Lichtenstein and Michael Sipser. Go is polynomial-space hard. *Journal of the ACM (JACM)*, 27(2):393–401, 1980.

Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. *arXiv preprint arXiv:1802.10501*, 2018.

Laurent Orseau and Levi HS Lelis. Policy-guided heuristic search with guarantees. *arXiv preprint arXiv:2103.11505*, 2021.

Laurent Orseau, Levi HS Lelis, Tor Lattimore, and Théophane Weber. Single-agent policy tree search with guarantees. *arXiv preprint arXiv:1811.10928*, 2018.

Sébastien Racanière, Théophane Weber, David P Reichert, Lars Buesing, Arthur Guez, Danilo Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 5694–5705, 2017.

Andreas Sedlmeier, Thomas Gabor, Thomy Phan, Lenz Belzner, and Claudia Linnhoff-Popien. Uncertainty-based out-of-distribution classification in deep reinforcement learning. *arXiv preprint arXiv:2001.00496*, 2019.

Yaron Shoham and Gal Elidan. Solving sokoban with forward-backward reinforcement learning. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, pp. 191–193, 2021.

Joao P Marques Silva and Karem A Sakallah. Grasp—a new search algorithm for satisfiability. In *The Best of ICCAD*, pp. 73–89. Springer, 2003.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Joshua Taylor and Ian Parberry. Procedural generation of sokoban levels. In *Proceedings of the International North American Conference on Intelligent Games and Simulation*, pp. 5–12, 2011.